International Journal of Machine Consciousness
Vol. 3, No. 1 (2011) 1–17

DOI: 10.1142/S1793843011000595

World Scientific
www.worldscientific.com

TOWARDS MACHINE CONSCIOUSNESS: GROUNDING ABSTRACT MODELS AS π-PROCESSES

PIERRE BONZON

HEC, University of Lausanne, Lausanne,
CH-1015, Switzerland
pierre.bonzon@unil.ch

We present a two-level model of concurrent communicating systems (CCS) to serve as a basis for machine consciousness. A language implementing threads within logic programming is first introduced. This high-level framework allows for the definition of abstract processes that can be executed on a virtual machine. We then look for a possible grounding of these processes into the brain. Towards this end, we map abstract definitions (including logical expressions representing compiled knowledge) into a variant of the π-calculus. We illustrate this approach through a series of examples extending from a purely reactive behavior to patterns of consciousness.

Keywords: Computational correlates of consciousness; concurrent communicating systems; π-calculus.

1. Introduction: A Methodological Problem

Will machines ever be able to think and enjoy consciousness? This question has been debated by philosophers and scientists alike, but no definitive answer has been given yet. Actors from both camps are sometimes very critical about each other. In fact, the whole domain seems to suffer from a lack of methodology and disciplined activity. Talking about the research done in robotics and AI, Brooks [2001] once wrote "None of these fields is experimental (...). There is no control experiment to compare against". Ten years later, when confronted with the apparent failure of current research in artificial intelligence and artificial life, he went on to ask if something was not going wrong, and if one were not actually missing something fundamental and currently unimagined in the models. He then conjectured: "Perhaps other mathematical principles or notions, necessary to build good explanations of the details of evolution, cognition, consciousness or learning will be discovered and invented" [Brooks, 2001].

In our view, research in these subjects should be inspired by the methods of theoretical physics, i.e., in as much the same way as *mathematical* abstractions (without apparent relations with intuitive perceptions) are used to model the properties and interactions of elementary particles, *computational* abstractions

should be used to model the cognitive activity taking place in the brain. The problem would be then somehow "ground" these processes, i.e., to embody them into biological neural networks.

We further argue that the computational abstraction needed for this task is already available, namely *concurrent communicating systems* (*CCS*). What is lacking today is an effective way of using this powerful notion, which tends to be buried into opaque code (as too often in AI, "the theory is the program"), whereas it should be at the heart of the model's definition.

Towards this goal, we propose a two-level model of CCS. At the first level, resulting from the integration of communication into a model of intelligent agents [Bonzon, 2002a], processes are defined in a high level language that provides the necessary tool for the simulation of cognitive functions. At the second level, abstract definitions (including logical expressions representing compiled knowledge) are mapped into a variation of the π-calculus [Milner, 1999]. The basic functionality of this lower-language bears strong similarities with artificial neural networks: each of these two formalisms allows for the representation of interconnected units (e.g., neurons) whose configuration changes over time, and thus offers a possible way to ground computations into brain processes. As both the high-level and the lower-level language we introduce can be compiled and executed on a virtual *abstract machine* [Bonzon, 2002b], experiments can be easily reproduced. At the end, this should allow for a systematic and repeated exploration, as required by the scientific experimental method.

We come now to the very object of this exploration. The study of consciousness extends in many directions. At one end lies the so-called "search for the neural correlates of consciousness" [Atkinson *et al.*, 2000]. Briefly, this approach involves "isolating the neural processes that correlate with various states of consciousness". Following the increasing availability of brain imaging techniques, the experimental research conducted in this direction is flourishing. But this approach has failed so far to provide us with any accurate knowledge about neuro-physiological mechanisms. At the other end, we find theories that are formulated merely in terms of the information processing presumably conducted by the brain [Sun, 1999]. This line of research has often been described as constituting a complementary "search for the computational correlates of consciousness" [Cleeremans, 2005]. This second approach, however, has yet to produce experimental platforms. As reported in [Arrabales and Sanchis de Miguel, 2008], there hardly exist a few actual implementations that furthermore rely on metaphors (such as the Global Workspace model [Baars, 1988]) "that simply help us understand in a holistic way how the human mind works (…) far away from an established body of scientific knowledge".

Our first aim being precisely to give up metaphors in favor of computational abstractions, we shall try and delineate aspects of consciousness that could be amenable to computational processes. We will therefore not be concerned by the so-called <hard> problem of consciousness related to the phenomenology of feelings and sensations, and be content with the <easy> ones related to the awareness of one's own activity and perceptions. More precisely, we will try and model the functionality of

episodic memory [Tulving, 1999]. In short, this functionality allows for the thinking and deliberation we do perform consciously or the events that hit our consciousness to get somehow memorized and then later recalled. The abstract robot model that we will define in this paper works in a similar manner. This similarity will not allow us, however, to argue that it acts consciously. The only claim that we will make is that it does enjoy a property it shares with a conscious mind, i.e., that of using a pervasive tool identified with thoughts in order to keep and retrieve traces of its past activity.

Our model will also be in line with the idea that humans are not directly conscious of their thoughts but rely on that for intermediate representations and processes. As formulated by Crick and Koch [2000] when reporting about Jackendoff [1987] postulates: "what is conscious about thoughts is visual or other images, or talking to oneself. (…) visual and verbal images are associated with intermediate-level sensory representations, which are in turn generated from thoughts by the fast processing mechanisms in short-term memory. Both the process of thought and its content are not directly accessible to awareness". In order to account for this intermediation, or in other words to allow for the indirect access to thoughts via intermediate-level representations, some kind of communication must take place. We do thus postulate in turn that if intermediate representations are the support of conscious thoughts, then communication is their trigger. More precisely, this new postulate can be formulated as follows: when humans think, they basically engage in synchronized communication of some internal representations.

We choose to introduce our tools in context, i.e., by illustrative examples. Following a developmental approach, we shall start with a model of purely reactive behavior. We then show how it can be extended into a more complex model incorporating a deliberative behavior. We further introduce a kind of awareness into the system by giving it the ability to keep traces of its deliberation processes, thus ending up with a model exhibiting the functionality of episodic memory. Conscious thinking can thus be bound to a particular type of activity, i.e., *deliberation* involving the processing of internal representations identified with a model of thoughts.

Finally, to validate our developments, we present a model simulating the first level of animal consciousness according to the hierarchy of Pepperberg and Lynn [2000].

2. A Test Bed for the Simulation of a Simple Robot

We shall base our developments on a simple robot model to be used as a test bed for experiments. Let us consider any kind of autonomous vehicle or mobile robot that can move towards a target to perform a single task (from simply picking up a ball to fixing a problem). For the purpose of a simulation, we assume that the robot is equipped with physical sensors allowing for the detection of both its position and that of targets. We shall further assume that the robot's behavior can be defined in terms of plans, each plan consisting in turn of actions. As the robot moves and acts, its current plan and action are selected using fixed rules as well as data obtained from its sensors.

2.1. *Rules for the selection of the robot's current plan and action*

In this simple model, only one of two possible plans *go* or *return* can be selected according to the following *plan* selection rules:

(1) "if there is a *target* to be served, then select plan *go*"
(2) "if there is no *target* to be served, then select plan *return*"

The choice of an action in a plan follows from *do* selection rules:

(3) "when using plan *go*, if moving towards the *target*, then select action *forward*"
(4) "when using plan *go*, if reaching the *target*, then select action *work*"
(5) "when using plan *return*, if moving towards *home*, then select action *backward*"
(6) "when using plan *return*, if reaching *home*, then select action *rest*".

2.2. *First-order logical representation of the robot's selection rules*

In order to represent the robot's behavior in logical terms, let us first consider the following predicates:

target(x) meaning "a *target* is located at position x"
at(x) meaning "the robot is *at* position x"
home(x) meaning "the *home* location to return to is at position x"
plan(p) meaning "plan p is selected"
do(p,a) meaning "within plan p, action a is selected".

The robot's selection rules can then be represented by the following logical implications (in accordance with the Prolog syntax, conjunctions are represented by the operator " , " and variables start with capital letters):

```
1)   target(_)                  →   plan(go)
2)   not target(_)              →   plan(return)
3)   (at(X), not target(X))     →   do(go,forward)
4)   (at(X),     target(X))     →   do(go,work)
5)   (at(X), not home(X))       →   do(return,backward)
6)   (at(X),     home(X))       →   do(return,rest)
```

These implications, which will be used to drive the robot's behavior and stored as fixed data, are called *compiled knowledge.*

3. A Language for Concurrent Communicating Processes

Implementing a simulation of a simple robot like the one defined above can be done in a variety of ways. In any case, there is no need to introduce communication at this stage. In order to allow for later developments, we shall nevertheless use at once a language allowing for the expression of communicating concurrent processes. Towards this end, we shall rely on our previous work that led us to define and implement such a language within the framework of logical programming [Bonzon, 2002a,b].

```
thread(sensor,
[P,A],
[(plan(P),
  do(P,A)| [effector(A)])])
```

Fig. 1. A logical sensor implementing reactive behavior.

3.1. *A first example: A logical sensor implementing reactive behavior*

Let us give first a concrete example of the use of this language. More precisely, consider the expression given in Fig. 1, which actually represents the definition of a *logical sensor* (as opposed to the physical sensors considered in the previous section) implementing a *reactive* behavior.

The name of a thread, i.e., `sensor` is first declared, together with two local variables, i.e., `P` and `A`. This is followed by a so-called *guarded command* (implementing a logical implication) consisting of a *guard*, i.e., plan `(P)`, `do(P,A)`, separated by the operator " | " from the *command* itself, i.e., `effector(A)`.

When implemented as a concurrent process, the above guarded command will allow for repeated *reaction* cycles defined as follows:

– select a *plan* `P` and an *action* `A` from this plan using *compiled knowledge*
– deliver the task of executing *action* `A` to an appropriate *effector*.

Note: In computer science, a thread is simply a sequence of instructions which may execute in parallel with other threads. With regard to a process, one may add that <the implementation of threads and processes differs from one operating system to another, but in most cases, a thread is contained inside a process. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources> (quoted from Wikipedia).

3.2. *Formal language definition*

We come now to a presentation of the complete language. We do not reproduce here the usual syntax for first-order expressions, the only divergence being that variable names must start with capital letters. The remaining formal syntax is defined by the grammar given in Fig. 2, where multiple choices are separated by the meta-operator " || ".

Each thread consists of a tree structure whose sequences contain messages separated by the conjunctive operator " , " and end with alternatives containing guarded messages separated by a disjunctive operator " ; ". Possible messages consist simply of the *tell/ask* pair used for communication, the *start/end* pair for creating and deleting a thread and the *effector* command allowing for the execution of actions by external actuators. Each thread is reentrant, i.e., is automatically resumed at the end of each alternative.

```
<thread>      ::= thread(<threadName>(<threadParams>),<varList>,<mesTree>)
<varList>     ::= [] || [<varName>|<varList>]
<mesTree>     ::= [] || <seq> || [<alt>]
<seq>         ::= [<mes>|<mesTree>]
<alt>         ::= <guardMes> || (<guardMes>;<alt>)
<guardMes>    ::= (<guard>|<mesTree>) || <mesTree>
<mes>         ::= <messageName>(<messageParams>)
<messageName> ::= tell || ask || start || end || effector
```

Fig. 2. BNF productions for agent threads.

4. Simulating Deliberative + Reactive Behaviors

In the second step of our test bed, we introduce a deliberative behavior. More precisely, the selection of a plan `P`, which in our previous model was part of the reactive process leading to action execution, now involves a *deliberative* preprocessing taking place before the action choice.

4.1. *Threads implementing a deliberative + reactive behavior*

The overall robot behavior can be expressed by the threads given in Fig. 3.

As it can be seen, `sensor(P)` threads are dependent on a plan `P` and have been extended with a deliberation process using `deliberate` compiled knowledge leading to choose and execute a *mental* action `B`.

4.2. *Running a simulation*

The `deliberate` compiled knowledge needed in this case is given by the following implications:

```
    target(_)  →  deliberate((end(sensor(return)),
                              start(sensor(go))))
not target(_)  →  deliberate((end(sensor(go)),
                              start(sensor(return))))
```

In other words, whenever there is a target to be served, the deliberation leads first to end the thread running a `return` plan (if there is any) and then starts a thread running the plan `go`, and vice versa.

```
thread(sensor(P),
[A,B],
[(deliberate(B)| [effector(B),
                  (do(P,A)| [effector(A)])])])
```

Fig. 3. Threads implementing a reactive + deliberative behavior.

```
thread(sensor(P),
[A,B],
[(do(P,A)| [effector(A),
            (deliberate(do(P,A),B)| [effector(B)])])]
```

Fig. 4. Threads implementing an activity-based reactive + deliberative behavior.

Deliberations could similarly be based on the robot's own activity, e.g., could lead him to return home whenever he has completed working on a target. In this case, deliberation would take place *after* the action choice, and depend on it, as shown in the sensor definition given in Fig. 4.

5. Introducing Conscious Recalls

In this last step, we model a form of consciousness taking place when recalls arise from *episodic memory*. As discussed in the Introduction, this functionality refers to the capability of remembering past experiences. Clearly, this functionality operates and/or occurs in conjunction with A-consciousness, i.e., with the ability to access mental contents. As a typical example, whenever one gets suddenly conscious of having done something wrong, a new event or a perception (e.g., being told of the negative effect of an action) and a recall from episodic memory (i.e., remembering having done it) are concurrently involved. Our postulate here will be that conscious recall is based on the *coincidental delivery* to the same locus of two or more internal communications. Translated into our model, this means that a foreground thread *recall* will be addressed concurrently by two or more background *server* threads, one acting as a *triggering* event and the other one as a *memory* store.

5.1. *A set of threads implementing conscious recalls*

Our simulation of conscious recalls results from an extension of our previous model given in Fig. 3. As sketched above, a new *recall* thread intended as the locus of consciousness communicates concurrently with two *server* threads, one allowing for the memorization of past events, and the other one acting as a trigger for the *recall* of memorized items. These two background server threads are started within a *reflective* post-processing taking place after the action execution. We thus get the new set of threads given in Fig. 5.

5.2. *Running a simulation*

The `reflect` compiled knowledge needed here is given by the following implication and fact:

```
at(X) → reflect(do(go,work), start(server(memory(work(X)))))
reflect(do(return,rest), start(server(trigger(memory))))
```

```
thread(sensor(P),
[A,B,C],
[(deliberate(B)|[effector(B),
                 (do(P,A)|[effector(A),
                           (reflect(do(P,A),C)|[effector(C)])])])])

thread(server(P),
[X],
[tell(X)])

thread(recall,
[F,P],
[ask(server(trigger(F))),
 ask(server(F(P))),
 effector(report(recall(P)))])
```

Fig. 5. A set of threads implementing conscious recalls.

In this simple case, the robot is asked to memorize the position of a target while it is working, and then to trigger a recall while it is resting.

Note: Arguments of *tell/ask* messages are unified with thread names. As an example, when thread `server(trigger(memory))` addresses thread `recall` via message `tell(X)`, then `X` gets bound to `recall`. Reciprocally when thread `recall` is addressing `server(trigger(memory))` via message `ask(server(trigger(F)))`, then `F` gets bound to `memory`.

6. Towards Grounding Abstract Processes into the Brain

In this section, we turn to the possible grounding of our models into biological neural networks. To understand what could possibly be achieved in pursuing such a goal, let us quote Van der Velde and de Kamps [2006]: "We know that the brain produces cognition (it is the only example we are certain of). So, if we have an idea of how computational processes could be matched onto brain processes, we could also get a clearer view of what could be missing in the computational account".

We first note the analogy that does exist between the syntactic properties of our model and the topology of synaptic connections:

— As shown by the formal syntax given in Sec. 3, threads grow as trees by pushing branches, these branches being connected to the branches of other threads through *tell/ask* communication primitives (we ignore for the moment the possibly for threads to extend sidewise by starting new threads).
— Neurons do grow like trees as well: Around the cell body is a branching dendritic tree that receives signals sent from other neurons through their axon, the axon itself possibly terminating in a branching synaptic tree (we restrict ourselves here to axonal chemical synapses, ignoring for the moment the sidewise dendritic gap junctions, to which we shall turn in later developments), see Fig. 6.

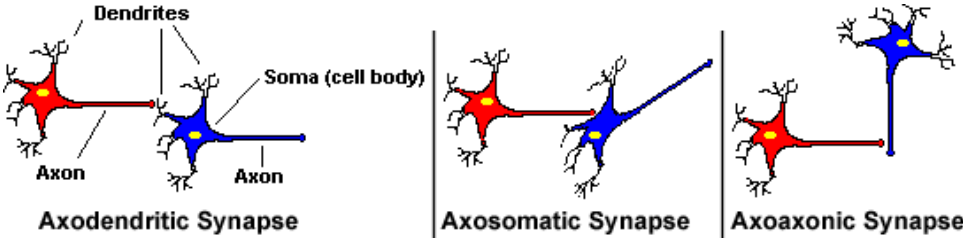

Fig. 6. Pairs of neurons with various axonal synaptic connections.[1]

Sets of interconnected neurons, or neural nets, form a structure in which micro-currents propagate as neurons <integrate and fire>. As it is known from recent advances in the theory of concurrent parallel processes [Milner, 1999], this continuous flow of signal transmissions is akin to the dynamic reconfiguration of a communication network. A powerful tool standing as a universal model of computation, namely the *π-calculus*, has been designed for expressing concurrent processes whose configuration may change during the computation. The basic idea behind the π-calculus (which, similarly to recurrent neural networks, has been shown to be Turing equivalent, i.e., capable of calculating any computable function) is that in order to model any communication system, one only needs to consider channels with given names, and then to pass these names over them. Towards this end, two communications primitives are defined, namely the *send(x,y)/receive(x,y)* pair allowing for the sending/receiving of name y on channel x.

Similarities do exist between the π-calculus and at least one model of neural-symbolic integration. Actually, the temporal synchrony mechanism that lies at the heart of the architecture proposed by Shastri and Ajjanagadde [1993] is closely related to the operational principles of the π-calculus: by slightly extending the facilities offered at the π-primitive level, it is indeed possible to rewrite their examples as π-processes.

We therefore postulate that a possible way of grounding behaviors into the brain is to try and work simultaneously on two fronts, i.e., to define and implement both neural nets and models of thoughts in terms of the π-calculus.

6.1. *Implementing neural nets as π-processes*

The original π-calculus is so minimal that it does not contain primitives such as numbers, booleans, variables, or the usual flow control statements such as "if... then...else". We can actually look at the π-calculus as a trimmed-down variant of our language for threads, the basic differences being that:

— Communication primitives of the π-calculus, i.e., *send/receive*, have two arguments;
— Logic formulas cannot be interpreted directly in the π-calculus.

[1] Used with permission of Eric H. Chudler, Ph.D., Neuroscience for Kids, http://faculty.washington.edu/chudler/neurok.html.

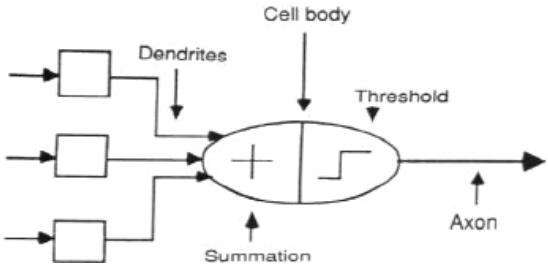

Fig. 7. A simple artificial neuron.[2]

```
process(body,
 [X1,X2,X3,Summation,Threshold],
 [receive(dendrite1,X1),
  receive(dendrite2,X2),
  receive(dendrite3,X3),
  (Summation is X1+X2+X3,
   Summation >= Threshold | [send(axon,Summation)])])
```

Fig. 8. A simple artificial neuron represented as a π-process.

In order to get an effective programming tool, we have developed an extension of the π-*calculus* that uses guarded commands, making extended π-*processes* look similar to our threads and executable on our virtual machine. As an example, let us consider the schema given in Fig. 7 describing the basic functioning of a neuron.

This simple artificial neuron can be represented by the extended π-process given in Fig. 8.

Both *neural nets* and π-*processes* are systems of interconnected units whose wired configuration changes over time (with <wired> referring here to <active> connections). In the case of neural nets, active connections result from a firing neuron sending a signal along its axon onto other neurons (as indicated above, we refrain for the moment from considering sidewise connections through dendritic gap junctions). In the case of π-*processes*, active connections are determined by *send/receive* commands.

6.2. *Implementing behaviors as π-processes*

The translation of threads into π-processes requires that compiled knowledge given by logical expressions be "flattened" and passed over on named channels (as noted above, this is very similar to the proposal put forward by Shastri and Ajjanagadde [1993] in the context of symbolic-neural integration). Starting with the model of reactive behavior given in Sec. 3.1, but without yet introducing plans, we get the π-processes given in Fig. 9.

[2] *Source*: Neural Networks, Chr. Stergiou and D. Siganos, Imperial College of London, http://www.doc.ic.ac.uk/~nd/surprise96/journal/vol4/cs11/report.html.

```
process(forward,
[X,Y],
[receive(target,X),
 receive(at,Y),
 (X\=Y | [effector(forward)])])

process(work,
[X,Y],
[receive(target,X),
 receive(at,Y),
 (X=Y | [effector(work)])])

process(backward,
[X,Y],
[receive(not target,_)
 receive(home,X),
 receive(at,Y),
 (X\=Y | [effector(backward)])])

process(rest,
[X,Y],
[receive(not target,_),
 receive(home,X),
 receive(at,Y),
 (X=Y | [effector(rest)])])
```

Fig. 9. Reactive behavior as π-processes.

As it can be seen in these various processes, the communication taking place on channels *target*, *not target*, *at* and *home* actually replicates the *reactive* compiled knowledge given in Sec. 2.2. To allow for the sidewise connections needed for implementing the deliberation presented in Sec. 4, let us introduce generic *server* processes that can be started and ended from within any regular process:

```
server(F(P),
[],
[send(F,P)])
```

Plans can be then implemented as background servers whose communication with action processes will replicate the implications given in Sec. 4.2, e.g.:

```
process(go,
[],
[receive(target,_),
  end(plan(return)),
  start(plan(go))])
process(forward,
[X,Y],
[receive(plan,go),
    (continued as before) ...])
```

Coming finally to the model of conscious recall given in Sec. 5, the processes corresponding to actions *work* and *rest* must be first extended to start a background *server* memorizing the position of a target while working and to trigger a foreground *recall* while resting, thus replicating the *reflect* compiled knowledge given in Sec. 5.2:

```
process(work,
[X,Y],
[receive(plan,go),
 receive(target,X),
 receive(at,Y),
 (X = Y | [effector(work),
           start(memory(work(X)))])])
process(rest,
[X,Y],
[receive(plan,return),
 receive(home,X),
 receive(at,Y),
 (X = Y | [effector(rest),
           start(trigger(memory))])])
```

The foreground *recall* process can be then defined as follows:

```
process(recall,
[F,P],
[receive(trigger,F)),
 receive(F,P),
 effector(report(recall(P)))])
```

To summarize, computations taking place in reactive models develop within axonal-dendritic networks via *send/receive* operations. In contrast, computations taking place in deliberative models develop *sidewise* using background servers, which in turn are coupled with foreground processes via *send/receive* operations, thus representing the equivalent of dendritic-dendritic gap junctions which can be started/ended "on the fly" via start/end operations.

7. Case Study: Modeling the First Level of Animal Awareness

In this last section, we do some steps towards the validation of our theoretical developments in the context of animal consciousness. Relying on a study by Pepperberg and Lynn [2000], we present a simulation of an experiment putting to light what these authors suggest might be the first level of animal awareness.

In this experiment, pigeons can peck either a red or a green button, and are confronted at the same time with a red sample [Zentall *et al.*, 1981]. Green being actually the good choice (i.e., pecking the green button rewards them with food), after awhile pigeons have learned to <choose non-match>, i.e., they have become *aware* of the good choice and thus consistently peck the green button.

Similarly to our previous example involving a robot, we shall distinguish three levels in our simulation. At the first level, which can be said to correspond to reflexes (either innate or learned), actions are defined within plans. At the second level, deliberation leads to plan selection. Finally at the third level, reflection triggers consciousness.

Pigeons have two possible plans to choose from, i.e., *random* and *avoid*, and there is only one action *peck(x)*, defined selectively as follows:

— "when using plan *random*, then randomly *peck* anyone of the two buttons";
— "when using plan *avoid*, then *peck* the button that does *not match* the sample".

Let us now consider the following predicates:

choices(l)	meaning	"l is the list of possible choices"
random(x,l)	meaning	"x is a value randomly selected from the list l"
member(x,l)	meaning	"x is any value belonging to the list l"
sample(x)	meaning	"x is the value corresponding to the sample"
do(p,a)	meaning	"within plan p, action a is selected".

The *peck* action can then be defined by the following implications:

```
(choices(L),
 random(X,L))     →   do(random,peck(X))
(choices(L),
 member(X,L),
 not sample(X))   →   do(avoid,peck(X))
```

Plan *random* is always selected by default. In order for the pigeons to learn to choose the *avoid* plan, we shall postulate that as a result of their sensory experiences, they do engage in two deliberative activities *see* and *sort* defined as follows:

```
sample(X)       →   see(peck(X),
                        start(server(match(X))))
not reward(X)   →   sort(peck(X),
                        start(server(reject(X))))
```

In other words, when pecking a button that does *match* the sample, pigeons start a server ***match(X)*** intended to communicate with a *learning* task (defined below in thread ***learn***). Similarly, when not rewarded with food, they start a server ***reject(X)***. Finally, a *reflective* step acting as a consciousness trigger will lead them to adopt the ***avoid*** plan as soon as a given learning *threshold* is reached:

```
threshold((match,reject)(X)) → reflect((start(server(match(X))),
                                          start(server(reject(X)))),
                                        (end(sensor(random)),
                                          start(sensor(avoid))))
```

```
thread(sensor(P),
[A,B,C,D],
[(do(P,A) | [effector(A),
             (see(A,B),
              sort(A,C) | [effector(B),
                           effector(C),
                           (reflect((B,C),D)|[effector(D)])])])])
thread(server(P),
[X],
[tell(X)])

thread(learn((F,G)(X)),
[],
[ask(server(F(X))),
 ask(server(G(X))),
 effector(increment((F,G)(X))),
 end(server(F(X))),
 end(server(G(X)))])
```

Fig. 10. A set of threads implementing the first level of animal awareness.

The overall process can now be defined as follows, where the effector ***increment*** acts as a *counter* to be tested using the ***threshold*** predicate given in Fig. 10.

Note: Switching to the ***avoid*** plan accounts for the pigeons new behavior. Our simulation, which does rely on compiled knowledge (i.e., the definition of the *peck* action under the form of implications), does not provide any clue of how the new reflex is actually built into the pigeon's brain. In order to simulate this process, action definitions could be represented as threads, and their adaptation made explicit through communication with the learning process.

Our simulation clearly follows Zentall *et al.* [1981] findings, i.e., that pigeons do not learn to choose green, but learn to match and avoid pecking the match. As a matter of fact, as pointed out by Pepperberg and Lynn's, < a pigeon need not be aware of anything specific about the green sample, nor that is it being trained with respect to oddity, nor of any concept of redness >. One might then rightfully ask: Where does awareness come into the picture? Pepperberg and Lynn's answer is that pigeons have the first level of awareness: < the ability to follow a simple rule involving perception of a specific item and its avoidance >. As these authors further note, this ability does not necessarily mean < being aware of devising or following the rule >. They go on suggesting that an organism has a second level of awareness when it < knows not only that a rule exists with respect to "likeness/ familiarity" (first level), but is aware enough of the rule to transfer it across situation >. This is not the case for pigeons that do not transfer easily to a task where red is correct.

8. Related Work

As already stated in the Introduction, there is not much work done along the same lines as ours to report about. Existing proposals generally stop short of any implementation, and thus do not allow for the kind of experimentation we call for. According to the latest survey [Cleeremans, 2005], the search for the computational correlates of consciousness tends so far to concentrate on theoretical aspects, and does not seem to have led yet to possible groundings into brain processes. A notable exception is given by the recent model put forward by Hameroff [2009]. His framework distinguishes two kinds of coordinated activity presumably taking place within the brain, one being identified with the "neuro-computation" in axonal−dendritic synaptic networks of "integrate-and-fire" neurons, and the other one deriving from neuronal groups linked by dendritic−dendritic gap junctions. While the first mechanism is credited with the non-conscious cognitive brain functions (including sensory processing and control of behavior), the second one is supposed to enable collective integration and volitional choices correlating with consciousness. The strong analogies existing between Hameroff's < conscious > pilot located in < dendritic webs >, on one hand, and our model of deliberation and conscious recalls which develops sidewise via background processes/threads, on the other, is rather striking. While developed independently and through quite different means, they do concur to propose a plausible explanation for combined computational and neural correlates of consciousness.

Our computational architecture looks compatible too with the requirements and/or consequences of Edelman's theory of *neuronal Darwinism*. Some of the salient features of this theory can be well illustrated through the following quote [Edelman, 1987]: "the brain is dynamically organized into cellular populations containing individually variant networks (…). Coherent temporal correlations of the responses of sensory receptors sheets, motor ensembles, and interacting neuronal groups in different brain regions occur by means of re-entrant signaling. Such signaling is based on the existence of reciprocally connected neural maps". Further investigations are clearly needed in order to establish a firm link between the two approaches, but it is worth mentioning here the obvious similarities that do exist at first sight between Edelman's rather intuitive concepts of *variant networks* and *reciprocally connected neural maps*, on one hand, and our own formal presentation of networks involving various neuronal connections implemented via extended π-processes, on the other.

9. Conclusion

As a possible basis for machine consciousness, we propose to adopt a two-level model of *concurrent communicating systems (CCS)*. At the first level, processes can be defined in a high-level language resulting from the integration of communication into a model of intelligent agents. At the second level, these abstract definitions can be mapped into a variation of the π-calculus, offering thus a possible way to ground

computations into brain processes. As both the high-level and the lower-level languages we introduce can be compiled and executed on a virtual *abstract machine*, experiments can be easily reproduced, allowing for a systematic and repeated exploration, as required by the scientific experimental method.

A definite distinctive feature of the abstract level we propose is the clear distinction it makes between *compiled knowledge*, on one hand, and *communication threads*, on the other. While compiled knowledge represents innate (or programmed) behavior, threads offer ways for

— processing this knowledge;
— delivering effector commands;
— acquiring learned behavior.

Being both the medium and the vehicle of deliberations leading to actions, threads can be identified with a model of *thoughts*. Therefore the claim we make here is that our abstract model does enjoy a property it shares with a conscious mind, namely that of using a pervasive tool identified with thoughts in order to keep and retrieve traces of its past activity.

As an illustration, we did implement both a simple abstract robot that is capable of recalling its past activity in response to a triggering event and a model for the first level of animal consciousness. As already stated in our Introduction, we do not claim to have thus succeeded in simulating consciousness, conscious behavior, or whatsoever. Clearly, any attempt at simulating consciousness should involve a real situated robot, possibly confronted with its own survival, etc. In any case, our aim was not to try such an attempt, but rather to propose, at the same time, both a test bed and a particular approach for implementing experiments.

Our test bed has been implemented and is running on a Prolog platform. Various extensions could be easily brought to enhance

— the virtual machine, in order to include more realistic perceptions;
— the definition of the abstract robot test bed.

It is worth noting here how our developmental approach correlates with evolutionist views: starting with the simplest of models defined by a single thread, we either defined new threads or enlarged existing ones, always keeping their kernel untouched and adding new functionalities at their interface. It is easy to imagine how this process could be iterated and the corresponding models get more and more complex until they eventually end up mimicking a brain. One could then ultimately invoke Brook's motto [2001], i.e., "the behaviors are the building blocks, and the functionality is emergent" to claim that the model "almost naturally" evolved to simulate consciousness. However, according to common sense, it is only when the eyes (or for that matter, any other sense) hit the mind that true consciousness really arises. Consequently, multiplying the number of seniors-motor controls, modeling

their interaction and eventually attempting a coupling with a real situated robot would be the next thing to do in order to go beyond simulations.

References

Arrabales M. R. and Sanchis de Miguel, A. [2008] "Applying machine consciousness models in autonomous situated agents," *Pattern Recognition Letters* **29**(8), 1033−1038.

Atkinson, A. P., Thomas, M. S. C. and Cleeremans, A. [2000] "Consciousness: Mapping the theoretical landscape," *Trends in Cognitive Sciences* **4**(10).

Baars, B. [1988] *A Cognitive Theory of Consciousness* (Cambridge Univ. Press).

Bonzon, P. [2002a] "An abstract machine for classes of communicating agents based on deduction," in *Intelligent Agents*, eds. Meyer, J. J. and Tambe, M. VIII, LNAI **2333**, Springer.

Bonzon, P. [2002b] "Compiling dynamic agent conversations," in *Advances in Artificial Intelligence*, eds. Jarke, M., Koehler, J. and Lakemeyer, G., LNAI **2479**, Springer.

Brooks, J. [1991] "Integrated systems based on behavior," *SIGART Bulletin* **2**(4).

Brooks, R. [2001] "The relationship between matter and life," *Nature* **401**.

Cleeremans, A. [2005] "Computational correlates of consciousness," *Progress in Brain Research* **150**, 81−98.

Crick, F. and Koch, Ch. [2000] "The unconscious homunculus," in ed. Metzinger, T., The Neuronal Correlates of Consciousness (MIT Press).

Edelman, G. [1987] *Neuronal Darwinism* (Basic Books Inc).

Hameroff, S. [2009] "The < conscious pilot > — dendritic synchrony moves through the brain to mediate consciousness," *Biological Physics*.

Jackendoff, R. [1987] *Consciousness and the Computational Mind* (MIT Press).

Milner, R. [1999] *Communicating and Mobile Systems: The π-Calculus* (Cambridge Univ. Press).

Pepperberg, I. and Lynn, S. [2000] "Possible levels of animal consciousness with reference to grey parrots (psittaccus erithacus)," *American Zoologist* **40**(6), 893−901.

Sun, R. [1999] "Computational models of consciousness: An evaluation," *J. Intelligent Systems* **9**.

Shastri, L. and Ajjanagadde, V. [1993] "From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony," *Behavioral and Brain Sciences* **16**(3), 417−451.

Tulving, E. [1999] "Episodic vs. semantic memory," in: *The MIT Encyclopedia of Cognitive Sciences*, eds. Keil, F. and Wilson, R., MIT Press.

Van der Velde, F. and de Kamps, M. [2006] "Neural blackboard architectures of combinatorial structures in cognition," *Behavioral and Brain Sciences* **29**, 37−108.

Zentall, T. *et al.* [1981] "Identity: The basis for both matching and oddity learning in pigeons," *Journal of Experimental Psychology: Animal Behavior Processes* **7**, 70−86.